

10/519489

**PROCESS FOR COMPILING AND EXECUTING SOFTWARE APPLICATIONS
IN A MULTI-PROCESSOR ENVIRONMENT**

BACKGROUND

5 The present invention relates to multi-application, secure operating systems for small, secure, external devices, such as smart card microcontrollers ("chips"). In particular, the present invention relates to mechanisms for secure runtime upload of applications onto such devices, authorisation mechanisms and the ability for authorised execution of multiple applications on the devices, where an applica-
10 tion may be potentially larger than the microcontroller memory size. The mechanism simplifies life-cycle smart card management aspects related to post-issuance chip application ("applet") upload and upgrade complexity. Also, mechanisms to prepare applications (i.e. compiler techniques) using a common set of project files in one compiler toolset, for execution in a multi-processor host & chip environment,
15 are described, thus automising the programming of the communication interfaces between the host and chip applications. An important motivation for the present invention is to provide a secure co-processor environment for general computer applications in order to counter software piracy, with accompanying development tools, and to allow new models for secure electronic software distribution and soft-
20 ware licensing. The present invention relates to US patent number 6,266,416, which is incorporated by reference herein. This patent describes a system for software license protection through the partial execution of a software application in a tamper-resistant external device.

 A number of technological challenges appear when a software application
25 must be split and executed on multiple processors, e.g. on a host computer workstation and an external token such as a smart card (chip card) microcontroller: The host application needs to be able to call functions residing on the token. Different variables now reside both on the host processor and the token. All functions which operate on (secure) variables residing on the token need to be executed on the
30 token. Variables need to be exchanged in both directions when (and only when) required.

All such runtime features which govern dual-processor execution need to be supported by development tools. Software protection is often an ad-hoc, post-development task. Therefore, the developer needs a *simple* and user friendly set of tools to determine what functions (or parts of code) to execute on the token, what programming variables shall reside on the token and what functions and programme variables shall reside on the host. The development tool needs to handle all protection aspects including usage of cryptographic algorithms. It should hide all low level communication protocol details from the developer, so that he/she does not have to care about design of PDUs (Programming Data Units) and hardware interface communication protocol programming.

Furthermore, the code executing on the token must be token hardware independent. Device-independent execution implies the availability of virtual machines and accompanying development compiler tools.

External devices in general do not have the same performance capacity as ordinary host processors. The more code which can be put onto the external device the better the security gets. Therefore, the solution must allow repeated and simple try-and-fail tuning of the protection by the developer in order to maximize performance and security.

The memory of a conventional smart card may be preloaded with a smart card application (commonly called applet). Java card(TM) is a smart card operating system defined by Sun Microsystems that is able to interpret a applets containing Java byte code. A Java card may store multiple applets which have different functionality. In many conventional smart cards, the applets are preinstalled into ROM and remain on the card forever. However, many smart card operating systems, including Java card, include means for uploading and storing applets in EEPROM, where applets may later be deleted. Other conventional smart cards include MULTOS and Smart Card for Windows.

U.S. Patent No. 5,923,884 (Peyret et al.) discloses a smart card that can store an entire application, including use rights, in its memory. In this smart card, the application is disposable. That is, the application can be removed once it is depleted, and can be replaced by a new applet. One suitable application for loading onto this smart card is a prepaid telephone time applet. Upon depletion of the time, a new applet with replenished use rights may be loaded to recharge the smart card. Alternatively, a completely different applet may be loaded.

Despite the flexibility of conventional smart cards, there are still deficiencies in such smart cards. For example, the size of the applet that executes in the smart card is limited by the memory of the smart card, particularly, the size of the non-volatile memory, which is typically EEPROM. Although non-volatile memory size in smart card continues to increase every year, it would be desirable if there were no memory constraints imposed on the size of the applets. Also, since non-volatile memory is expensive, it would be desirable to be able to execute an applet on a smart card which has less non-volatile memory than the size of even a small applet.

Furthermore, conventional smart cards upload applets in a static manner. That is, the applets are loaded into the smart card either at production time (e.g., preinstallation of the applet into ROM), as illustrated in Figure 1, or during a programming mode (e.g., scheme in U.S. Patent No. 5,923,884 (Peyret et al.)). So far there have been no dispositions (and apparently no requirements) to allow applet management (upload and/or deletion) to occur at the very moment while the applet itself is executing. Also, there have been no mechanisms described which allow the upload of only parts of an applet at the time. In patent GB 2163577, entire applications are uploaded, not fragments or independently executable blocks of software code. Current applet loading processes require separate management software which increases the time, complexity and expense associated with smart card usage. Global Platform (www.globalplatform.org) defines such standards for smart card management, including applet loading and deletion, which have been implemented by Java cards.

The traditional approach to application programming for multiple processors, e.g. for smart card applications which always operate in connection with an external host of some kind, is to develop and compile one application separately for each processor (as shown in Figure 7), taking care to define the communication interface between each processor very carefully. One problem with this traditional approach, is the necessity for application developers to carefully design (and at an early stage fix) the interfaces between the different processors. Currently there is no way a compiler operating on a smart card application alone, is able to modify programming interfaces between processors, without having the compiler for the host application doing the same modifications accordingly. In other words, in order to allow interfaces to be adapted and improved automatically by compiler

tools, the compilers (or compiler) need(s) to operate on both the host application source code and the smart card application source code simultaneously. However, in order for one compiler to operate simultaneously on two (or multiple) applications, the compiler needs to obtain information about what parts of the software application code belong to what processor platform. The present invention addresses these issues, and enables developer-friendly implementation of multiple-processor applications where the low-level PDU (Programming Data Unit) protocol implementational details are taken care of automatically by the compiler, and thus hidden for the developer.

SUMMARY OF THE INVENTION

To simplify applet management aspects it would be desirable if applets could be uploaded dynamically, at runtime, and without using separate management software. The present invention fulfils such unmet needs by allowing an applet to be transferred to an external unit (e.g., a smart card) for execution in the external unit at runtime without using separate management software, and by allowing the applet to be transferred to the smart card and executed in the smart card in sequentially transferred blocks of code, hereafter also referred as "QX blocks". ("QX" stems from the name of a smart card operating system developed by Sospita, and is an abbreviation for "seQure eXecution".)

The present invention allows for execution of software code of a software program on an external unit. The external unit is connected to a computer. The computer includes memory for holding the software code. The external unit includes input/output for communication with the computer, a processor, and memory.

At runtime of the software code, the software code is automatically uploaded to the memory of the external unit. The software code is then executed in the external unit using only the processor and the memory of the external unit.

In an additional feature of the present invention, the software code is arranged or parsed into a plurality of different blocks of code, wherein each block of code is independently executable. In this embodiment of the present invention, a first block of code required for execution is automatically uploaded to the memory of the external unit. Next, the first block of code is executed in the external unit using only the processor and the memory of the external unit. When required, subsequent blocks of code of the software code are then sequentially and automati-

cally uploaded and executed in the external unit. During this process, subsequent blocks of code overwrite previously uploaded blocks of code in the memory of the external unit if this is required to have enough memory in the external unit for the subsequent blocks of code.

5 In one preferred embodiment of the present invention, the software code is an applet, and each of the different blocks of code are QX blocks. The software program may include a plurality of applets interspersed within the software program. The process described above is applied to each applet.

10 In another preferred embodiment of the present invention, the software code includes a first portion and a second portion. The second portion is the software code that is executed in the external unit, and the first portion executes in the computer. After execution of the software code, the external unit sends back state information to the computer for subsequent use by at least the first portion of the software code.

15 The software code that executes in the external unit is preferably encrypted. If so, then each block of code must be decrypted in the external unit prior to execution.

The external unit is preferably a tamper-proof device, such as a smart card, USB token, PC card, a TEMPEST/tamper-protected workstation or a physically se-
20 cured computer server connected to a network.

Compiler development techniques are presented that allow one set of files to contain code both for the host application and for the chip application. Source code for a particular target processor is tagged, and during compilation transformed into encrypted (virtual) machine code. The tagged blocks of code are replaced by function calls which refer to the encrypted (virtual) machine code components, to the effect of allowing these blocks of protected code to be dynamically loaded onto the remote processor during runtime, decrypted, stored and executed.
25 The source code for a particular target processor is split into independently executing blocks of code, which allows the independent execution of each block of code, which in turn allows the target processor (with potentially limited available resources in terms of memory and CPU) to execute applications bigger than its available
30 memory.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing summary, as well as the following detailed description of preferred embodiments of the invention, will be better understood when read in conjunction with the appended drawings. For the purpose of illustrating the invention, there is shown in the drawings an embodiment that is presently preferred. It should be understood, however, that the invention is not limited to the precise arrangements and instrumentalities shown. In the drawings:

Figure 1: shows the prior art chip application upload and execution procedures. Applets are uploaded using a trusted application loader, prior to any execution of the applet itself or to the execution of its associated host application. Applets are typically loaded in secure sites before the smart cards are even issued, or within a secure site sometime after the smart card has been distributed to the customer. This procedure implies that the customer is required to bring his/her card to such a secure site in order to upgrade the card.

Figure 2: shows a preferred embodiment of the present invention, whereby the applet, or even just a part of the applet, is allowed to be securely uploaded onto the remote processor.

Figures 3 and 4: show the same configuration as Figure 2, but where time has elapsed and the remote processor contains state with regards to what applets and parts of applets are stored within.

Figure 5: shows a function call graph, showing interdependencies between four software functions.

Figure 6: shows the same function call graph as in Figure 5, additionally illustrating transitive dependencies of arguments to each of the functions.

Figure 7: shows prior art procedures for compiling multiple application for multiple processors using multiple compilers and development tools.

Figure 8: shows a dual-step sequential compilation procedure which allows one set of source files to be used to contain application code for multiple processors, and which allows the resulting executable of the same source files to embed (encrypted) runtime uploadable application code for each of the multiple processors.

Figure 9: Shows the advantage of incorporating tags as comment fields of traditional compilers, whereby traditional compilers can be used to generate code for simulation and debugging purposes.

Figure 10: shows an ANSI C programming language source code examples, where tags which identify processor code for a second processor have been incorporated.

Figure 11: shows the same code example of Figure 10, after compilation for the second processor. Notice that the code is still ANSI C programming language source code.

Figure 12: describes a cryptographic protocol for the secure transfer for a license (or a capability) from one secure processor to another, in a fashion which prevents duplication or piracy of the license.

10

DETAILED DESCRIPTION OF THE INVENTION

Certain terminology is used herein for convenience only and is not to be taken as a limitation on the present invention. In the drawings, the same reference letters are employed for designating the same elements throughout the several figures.

15

The present invention describes a mechanism in which chip applications (applets) are dynamically uploaded to the smart card during runtime, i.e. during the actual execution of the application itself. In effect, the chip application is bootstrapping itself potentially every time it, together with its corresponding host application, is executed. This approach is substantially different from all other smart card operating systems, where the smart card application is loaded onto the smart card in the production or post-production phase, but always before the execution of the actual chip application starts.

20

Figure 2 shows one preferred embodiment of the present invention. Computer 20 includes a memory 22 that contains software programs 10 and 11. The software program 10 includes a portion of unencrypted code 101 (i.e., the combined code labelled 101₁, 101₂ and 101₃) that executes on the computer 20. The software program 10 also includes chip application 102 (i.e. the combination of QX blocks 102₁ and 102₂). In this example, QX blocks 102₁ and 102₂ take up 10 Kilobytes and 15 Kilobytes memory respectively. Similarly, the software program 11 includes a portion of unencrypted code 111 (i.e., the combined code labelled 111₁, 111₂ and 111₃) that executes on the computer 20. The software program 11 also

30

includes chip application 112 (i.e. the combination of QX blocks 112₁ and 112₂). In this example, QX blocks 112₁ and 112₂ take up 4 Kilobytes and 9 Kilobytes memory respectively.

Figure 2 also shows an external unit 30 which is generally similar to the external units described above. The external unit includes CPU 36, input/output (I/O) 32, a chip application manager 34 and non-volatile memory 38, which usually is EEPROM, also might be flash memory or any other kind of memory, depending on the type of the external unit. The chip application manager 34 handles all management related to decryption and storage of chip applications and chip application blocks, before and during chip application runtime.

The application loader 24 and chip application (applet) manager 34 implements mechanisms to allow dynamic, runtime applet upload into the chip OS. As described above, each chip application is composed of a number of distinctly identifiable blocks (QX blocks). Every time a new QX block is to be executed, the application loader sends a request to the applet manager. In one embodiment of the present invention, the chip application manager first verifies that it contains a valid license for this QX block. If the license is valid, the applet manager continues to check if there is enough memory to store the QX block. If there is not enough space left, the applet manager selects "old" QX blocks which are allowed to be deleted, which collectively free enough space for the new QX block to be stored. The new QX block, which originally was stored encrypted inside the host executables, is transferred to the token, decrypted, its integrity verified, stored in memory and then finally it is invoked. This dynamic, runtime approach to applet management has several important advantages:

Smart card memory is no longer reserved for one application, but can be shared between all applets which associate with a valid license on the token. The memory is thus reused by multiple applets, allowing execution of more applets than there would otherwise be room for in the smart card memory.

Because the applet manager operates on QX blocks, and not just entire QX applets, the applet manager even allows one single QX applet which in itself is bigger than the available EEPROM to still be executed in the token.

Host executable files which embed QX applets provide an ideal storage and transport container for seamless and cost-efficient distribution of new applets as well as for applet upgrades, thus solving a majority of the expensive logistics of chip application lifecycle maintenance.

5 In the example of figures 2, 3 and 4, the chip memory 38 has 16 Kilobytes total of memory. In conventional smart card implementations the external unit 30 would not be capable of executing both chip application 102 or 112 together, because the total size of the chip applications (10K+15K=25K for chip application 102 and 4K+9K=13K for chip application 112) exceeds the size of the memory 38 (16K).
10 Furthermore, conventional smart card implementations the external unit 30 would not be capable of executing chip application 102 on its own, because the total size of chip application 102 (25K) alone exceeds the size of the memory 38 (16K).

The present invention provides a solution to this limitation. At compilation
15 time, each block of code is independently executable. During execution of software program 11 in Figure 2 (at time $t=1$), QX block 112₂ is transferred by the application loader 24 to the external unit 30. QX block 112₂ is decrypted and stored as a decrypted QX block 113₁ in memory 38. Because each block of code is independently executable, QX block 112₂ is allowed to be executed by the CPU 36. In another embodiment (not shown in the figure), QX block 112₂ is stored directly into
20 memory 38, without decryption. In yet another embodiment (not shown in figure), the Chip 30, upon reception of QX block 112₂, first verifies if it contains a capability to decrypt and execute this chip application. If so, QX block 112₂ is decrypted and stored as a decrypted QX block 113₁ in memory 38, and then executed by the
25 CPU 36.

At time $t=2$ during execution, QX block 112₂ needs to be executed. Figure 3 illustrates this scenario. Since memory 38 contains enough free space, QX block 112₂ is dynamically uploaded and stored together with the first QX block 112₁. Provided the chip memory 38 is large enough then in an alternative embodiment of
30 figures 2 and 3, QX block 112₁ and QX block 112₂ could be loaded onto the smart card memory at the same time $t=1$, just when the first QX block starts execution. At time $t=3$ during execution, the execution path of the software program reaches QX block 102₁, either because the host code 101 reaches here or because the chip application 102 does. If the chip application does, it sends a request to the

application loader 24 to instruct it to initiate upload of the new QX block. Now in either case the client host 20 sends a request to the chip 30 to initiate upload of the new QX block. The chip application manager 34 recognises that QX block 102₁ occupies 10Kilobytes memory, while the chip memory 38 is virtually full. In order to
5 allow upload and execution of QX block 102₁, the chip application manager 34 frees memory by deleting some of the QX blocks already stored on memory 38. In figure 4 the chip manager 34 determines that it is sufficient to delete QX block 103₁. It is deleted, and the new QX block 102₁ is uploaded to the chip, decrypted, stored and executed. Notice that in this scenario, both chip applications 102 and
10 112 are allowed to be active at the same time on the chip 30. Chip application 102 is at any time allowed to re-upload any QX blocks that are not present.

Before deleting a QX block, the applet manager 34 needs to determine what QX block or blocks to delete. Common deletion strategies exist for different systems, including selecting the "best fit" object memory size and required mem-
15 ory size, the least-used object or the least-recently-used object. In one embodiment of the present invention, each QX block may incorporate a "delete mode" attribute. The applet manager uses the attribute to determine what QX block(s) to delete next. Four delete levels are offered:

1) Delete on close: The QX block is automatically deleted every time the QX
20 applet terminates.

2) Keep on card; delete by anybody: After upload and execution the QX block remains on card until the applet manager, invoked by any QX applet, decides to remove it.

3) Keep on card; delete by self only: After upload and execution the QX
25 block remains on card until the applet manager, invoked by another QX block that belongs to the same QX applet, decides to remove it.

4) Keep on card: In this case the QX block is never deleted, unless perhaps by an authenticated request, e.g. using passwords. The password is set by the applet issuer, so normally only the issuer is able to delete the applet. This delete
30 mode resembles the traditional approach followed by Java cards, where applets are stored more or less statically on the card.

The mechanism for dynamic, runtime upload of QX blocks for multiple applets is repeated for any subsequent QX blocks until all QX blocks and all chip applications have been executed. Any resulting state parameter after execution of a

QX block is stored in the chip memory 38 or returned to the software program 22 for subsequent use by the software application.

As discussed above, one significant benefit of the present invention is that the software code to be executed in the external unit 30 is not constrained by the size of the memory 38 in the external unit 30. If an application program has a 50K applet, there can be ten sequential transfers of 5K of code. Other examples are within the scope of the present invention.

Software development for multiple processors is made complex due to the multiple communication interfaces which appear, threads of execution may be complex if concurrent programming is introduced etc. It is therefore of importance to develop automated tools to assist development as far as possible. Consider the following scenario: When software programs are executed in multiple processors, different parts of the software program are allocated to be executed on a specific processor. Furthermore, different program variables are allocated to be stored on various, specific processors. E.g., in a dual processor environment with two processors "host" and "chip", the host application may call the chip function Func1(A). The parameter A is passed as a function argument from the host to the chip. Assume that the parameters A and B reside on the host, and assume that Func1(A) currently executing on the chip needs to call Func2(B). Now, even if both functions Func1(A) and Func2(B) are present on the chip, then because B only resides on the host, the chip operating system needs to send a request back to the host, for the host to transmit parameter B to the chip. This procedure has several disadvantages. From a security perspective information is revealed when the chip suddenly requests a new parameter. From a performance point of view, the communication between the chip and the host may be slow.

To amend the situation described above, Func1(A) could be redefined into Func1(A,B). In this way the chip is sure to always receive both parameter A and B when Func1 is called. Hence, if and when Func1 requires to call Func2, parameter B will be present. Figures 5 and 6 illustrate a similar case. In Figure 5, four functions Func1(A), Func2(B), Func3(B,C) and Func4(D) and their call pattern is shown. Figure 6 shows the transformed functions: Since Func2 calls Func3 and Func4, Func2 needs to incorporate parameters for Func3 and Func4. I.e. Func2(B) is transformed into Func2(B,C,D). Similarly, Func1 calls Func2 and

Func3, so Func1(A) is transformed into Func1(A,B,C,D). Func3() calls no other functions, so no transformation is required. FuncD calls itself recursively. Since recursion does not add any new arguments, FuncD needs no transformation.

The task of redefining functions can be done manually by developers. This may be a tedious task, so the ideal solution is to let development tools, e.g. compilers, do the work automatically.

The traditional approach to program an application for multiple processors, e.g. for smart card applications which always operate in connection with an external host of some kind, is to develop one application separately for each processor, taking care to define the communication interface between each processor very carefully. One problem with this traditional approach, is exactly related to the problem above: There is no way a compiler operating on a smart card application alone, can (or even should) be able to modify the function interfaces, without having the compiler for the host application doing the same modifications accordingly. In other words, in order to allow interfaces to be adapted and improved automatically by compiler tools, the compilers (or compiler) need(s) to operate on both the host application source code and the smart card application source code simultaneously. However, in order for one compiler to operate simultaneously on two (or multiple) applications, the compiler needs to obtain information about what parts of the software application code belong to what processor platform. One solution could be to keep code for the different platforms in different files, and tell the compiler what files belong to what processor. This solution works fairly well, but is not optimal because the logical structure of the application is obscured when functions which belong together from a logical point of view are forcedly split apart to enable the compiler to associate the code with a particular platform.

An intelligent way to distinguish one code from the other. is to provide syntactical means to identify and mark up code for the various platforms. Recalling from above that a chip application (an applet) is divided into functionally independent QX blocks, a pair of uniquely identifiable keywords, e.g. QXBegin and QXEnd, can be used to mark the beginning and the end of one QX block. The keywords may be used repeatedly, so that in principle each new marked portion of the code constitutes a new unique QX block. The sum of all marked QX blocks constitutes the applet, or the chip application. All code which is not marked, i.e. all the "negative" blocks of code (from QXEnd to QXBegin), constitute the host applica-

tion. The markup language can trivially be extended for multiple (more than 2) application platforms, or blocks, of code that collectively constitute the applet. Every QX block is allocated a unique QXBlockId. Figure 10 illustrates the principle of marking up a software program for two processors; host and (smart card) chip.
5 All code which is present between QXBegin and QXEnd is allocated for chip execution.

A QX block is comparable to a Java card method. One significant difference between a Java card method and the QX blocks illustrated in Figure 10, are that Java card methods (and the entire Java card application) never is contained together with the host application code. Another important difference is that a QX block need not be an entire function. In fact, Figure 10 shows two QX blocks, neither of which is a definition of an entire function. In other words, there is nothing which restricts marked-up code (QX blocks) from being anything from a single-line statement, to multiple statements within a function (e.g. an ANSI C block, as determined by curly brackets "{" and "}"
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
105
110
115
120
125
130
135
140
145
150
155
160
165
170
175
180
185
190
195
200
205
210
215
220
225
230
235
240
245
250
255
260
265
270
275
280
285
290
295
300
305
310
315
320
325
330
335
340
345
350
355
360
365
370
375
380
385
390
395
400
405
410
415
420
425
430
435
440
445
450
455
460
465
470
475
480
485
490
495
500
505
510
515
520
525
530
535
540
545
550
555
560
565
570
575
580
585
590
595
600
605
610
615
620
625
630
635
640
645
650
655
660
665
670
675
680
685
690
695
700
705
710
715
720
725
730
735
740
745
750
755
760
765
770
775
780
785
790
795
800
805
810
815
820
825
830
835
840
845
850
855
860
865
870
875
880
885
890
895
900
905
910
915
920
925
930
935
940
945
950
955
960
965
970
975
980
985
990
995
1000
1005
1010
1015
1020
1025
1030
1035
1040
1045
1050
1055
1060
1065
1070
1075
1080
1085
1090
1095
1100
1105
1110
1115
1120
1125
1130
1135
1140
1145
1150
1155
1160
1165
1170
1175
1180
1185
1190
1195
1200
1205
1210
1215
1220
1225
1230
1235
1240
1245
1250
1255
1260
1265
1270
1275
1280
1285
1290
1295
1300
1305
1310
1315
1320
1325
1330
1335
1340
1345
1350
1355
1360
1365
1370
1375
1380
1385
1390
1395
1400
1405
1410
1415
1420
1425
1430
1435
1440
1445
1450
1455
1460
1465
1470
1475
1480
1485
1490
1495
1500
1505
1510
1515
1520
1525
1530
1535
1540
1545
1550
1555
1560
1565
1570
1575
1580
1585
1590
1595
1600
1605
1610
1615
1620
1625
1630
1635
1640
1645
1650
1655
1660
1665
1670
1675
1680
1685
1690
1695
1700
1705
1710
1715
1720
1725
1730
1735
1740
1745
1750
1755
1760
1765
1770
1775
1780
1785
1790
1795
1800
1805
1810
1815
1820
1825
1830
1835
1840
1845
1850
1855
1860
1865
1870
1875
1880
1885
1890
1895
1900
1905
1910
1915
1920
1925
1930
1935
1940
1945
1950
1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
2020
2025
2030
2035
2040
2045
2050
2055
2060
2065
2070
2075
2080
2085
2090
2095
2100
2105
2110
2115
2120
2125
2130
2135
2140
2145
2150
2155
2160
2165
2170
2175
2180
2185
2190
2195
2200
2205
2210
2215
2220
2225
2230
2235
2240
2245
2250
2255
2260
2265
2270
2275
2280
2285
2290
2295
2300
2305
2310
2315
2320
2325
2330
2335
2340
2345
2350
2355
2360
2365
2370
2375
2380
2385
2390
2395
2400
2405
2410
2415
2420
2425
2430
2435
2440
2445
2450
2455
2460
2465
2470
2475
2480
2485
2490
2495
2500
2505
2510
2515
2520
2525
2530
2535
2540
2545
2550
2555
2560
2565
2570
2575
2580
2585
2590
2595
2600
2605
2610
2615
2620
2625
2630
2635
2640
2645
2650
2655
2660
2665
2670
2675
2680
2685
2690
2695
2700
2705
2710
2715
2720
2725
2730
2735
2740
2745
2750
2755
2760
2765
2770
2775
2780
2785
2790
2795
2800
2805
2810
2815
2820
2825
2830
2835
2840
2845
2850
2855
2860
2865
2870
2875
2880
2885
2890
2895
2900
2905
2910
2915
2920
2925
2930
2935
2940
2945
2950
2955
2960
2965
2970
2975
2980
2985
2990
2995
3000
3005
3010
3015
3020
3025
3030
3035
3040
3045
3050
3055
3060
3065
3070
3075
3080
3085
3090
3095
3100
3105
3110
3115
3120
3125
3130
3135
3140
3145
3150
3155
3160
3165
3170
3175
3180
3185
3190
3195
3200
3205
3210
3215
3220
3225
3230
3235
3240
3245
3250
3255
3260
3265
3270
3275
3280
3285
3290
3295
3300
3305
3310
3315
3320
3325
3330
3335
3340
3345
3350
3355
3360
3365
3370
3375
3380
3385
3390
3395
3400
3405
3410
3415
3420
3425
3430
3435
3440
3445
3450
3455
3460
3465
3470
3475
3480
3485
3490
3495
3500
3505
3510
3515
3520
3525
3530
3535
3540
3545
3550
3555
3560
3565
3570
3575
3580
3585
3590
3595
3600
3605
3610
3615
3620
3625
3630
3635
3640
3645
3650
3655
3660
3665
3670
3675
3680
3685
3690
3695
3700
3705
3710
3715
3720
3725
3730
3735
3740
3745
3750
3755
3760
3765
3770
3775
3780
3785
3790
3795
3800
3805
3810
3815
3820
3825
3830
3835
3840
3845
3850
3855
3860
3865
3870
3875
3880
3885
3890
3895
3900
3905
3910
3915
3920
3925
3930
3935
3940
3945
3950
3955
3960
3965
3970
3975
3980
3985
3990
3995
4000
4005
4010
4015
4020
4025
4030
4035
4040
4045
4050
4055
4060
4065
4070
4075
4080
4085
4090
4095
4100
4105
4110
4115
4120
4125
4130
4135
4140
4145
4150
4155
4160
4165
4170
4175
4180
4185
4190
4195
4200
4205
4210
4215
4220
4225
4230
4235
4240
4245
4250
4255
4260
4265
4270
4275
4280
4285
4290
4295
4300
4305
4310
4315
4320
4325
4330
4335
4340
4345
4350
4355
4360
4365
4370
4375
4380
4385
4390
4395
4400
4405
4410
4415
4420
4425
4430
4435
4440
4445
4450
4455
4460
4465
4470
4475
4480
4485
4490
4495
4500
4505
4510
4515
4520
4525
4530
4535
4540
4545
4550
4555
4560
4565
4570
4575
4580
4585
4590
4595
4600
4605
4610
4615
4620
4625
4630
4635
4640
4645
4650
4655
4660
4665
4670
4675
4680
4685
4690
4695
4700
4705
4710
4715
4720
4725
4730
4735
4740
4745
4750
4755
4760
4765
4770
4775
4780
4785
4790
4795
4800
4805
4810
4815
4820
4825
4830
4835
4840
4845
4850
4855
4860
4865
4870
4875
4880
4885
4890
4895
4900
4905
4910
4915
4920
4925
4930
4935
4940
4945
4950
4955
4960
4965
4970
4975
4980
4985
4990
4995
5000
5005
5010
5015
5020
5025
5030
5035
5040
5045
5050
5055
5060
5065
5070
5075
5080
5085
5090
5095
5100
5105
5110
5115
5120
5125
5130
5135
5140
5145
5150
5155
5160
5165
5170
5175
5180
5185
5190
5195
5200
5205
5210
5215
5220
5225
5230
5235
5240
5245
5250
5255
5260
5265
5270
5275
5280
5285
5290
5295
5300
5305
5310
5315
5320
5325
5330
5335
5340
5345
5350
5355
5360
5365
5370
5375
5380
5385
5390
5395
5400
5405
5410
5415
5420
5425
5430
5435
5440
5445
5450
5455
5460
5465
5470
5475
5480
5485
5490
5495
5500
5505
5510
5515
5520
5525
5530
5535
5540
5545
5550
5555
5560
5565
5570
5575
5580
5585
5590
5595
5600
5605
5610
5615
5620
5625
5630
5635
5640
5645
5650
5655
5660
5665
5670
5675
5680
5685
5690
5695
5700
5705
5710
5715
5720
5725
5730
5735
5740
5745
5750
5755
5760
5765
5770
5775
5780
5785
5790
5795
5800
5805
5810
5815
5820
5825
5830
5835
5840
5845
5850
5855
5860
5865
5870
5875
5880
5885
5890
5895
5900
5905
5910
5915
5920
5925
5930
5935
5940
5945
5950
5955
5960
5965
5970
5975
5980
5985
5990
5995
6000
6005
6010
6015
6020
6025
6030
6035
6040
6045
6050
6055
6060
6065
6070
6075
6080
6085
6090
6095
6100
6105
6110
6115
6120
6125
6130
6135
6140
6145
6150
6155
6160
6165
6170
6175
6180
6185
6190
6195
6200
6205
6210
6215
6220
6225
6230
6235
6240
6245
6250
6255
6260
6265
6270
6275
6280
6285
6290
6295
6300
6305
6310
6315
6320
6325
6330
6335
6340
6345
6350
6355
6360
6365
6370
6375
6380
6385
6390
6395
6400
6405
6410
6415
6420
6425
6430
6435
6440
6445
6450
6455
6460
6465
6470
6475
6480
6485
6490
6495
6500
6505
6510
6515
6520
6525
6530
6535
6540
6545
6550
6555
6560
6565
6570
6575
6580
6585
6590
6595
6600
6605
6610
6615
6620
6625
6630
6635
6640
6645
6650
6655
6660
6665
6670
6675
6680
6685
6690
6695
6700
6705
6710
6715
6720
6725
6730
6735
6740
6745
6750
6755
6760
6765
6770
6775
6780
6785
6790
6795
6800
6805
6810
6815
6820
6825
6830
6835
6840
6845
6850
6855
6860
6865
6870
6875
6880
6885
6890
6895
6900
6905
6910
6915
6920
6925
6930
6935
6940
6945
6950
6955
6960
6965
6970
6975
6980
6985
6990
6995
7000
7005
7010
7015
7020
7025
7030
7035
7040
7045
7050
7055
7060
7065
7070
7075
7080
7085
7090
7095
7100
7105
7110
7115
7120
7125
7130
7135
7140
7145
7150
7155
7160
7165
7170
7175
7180
7185
7190
7195
7200
7205
7210
7215
7220
7225
7230
7235
7240
7245
7250
7255
7260
7265
7270
7275
7280
7285
7290
7295
7300
7305
7310
7315
7320
7325
7330
7335
7340
7345
7350
7355
7360
7365
7370
7375
7380
7385
7390
7395
7400
7405
7410
7415
7420
7425
7430
7435
7440
7445
7450
7455
7460
7465
7470
7475
7480
7485
7490
7495
7500
7505
7510
7515
7520
7525
7530
7535
7540
7545
7550
7555
7560
7565
7570
7575
7580
7585
7590
7595
7600
7605
7610
7615
7620
7625
7630
7635
7640
7645
7650
7655
7660
7665
7670
7675
7680
7685
7690
7695
7700
7705
7710
7715
7720
7725
7730
7735
7740
7745
7750
7755
7760
7765
7770
7775
7780
7785
7790
7795
7800
7805
7810
7815
7820
7825
7830
7835
7840
7845
7850
7855
7860
7865
7870
7875
7880
7885
7890
7895
7900
7905
7910
7915
7920
7925
7930
7935
7940
7945
7950
7955
7960
7965
7970
7975
7980
7985
7990
7995
8000
8005
8010
8015
8020
8025
8030
8035
8040
8045
8050
8055
8060
8065
8070
8075
8080
8085
8090
8095
8100
8105
8110
8115
8120
8125
8130
8135
8140
8145
8150
8155
8160
8165
8170
8175
8180
8185
8190
8195
8200
8205
8210
8215
8220
8225
8230
8235
8240
8245
8250
8255
8260
8265
8270
8275
8280
8285
8290
8295
8300
8305
8310
8315
8320
8325
8330
8335
8340
8345
8350
8355
8360
8365
8370
8375
8380
8385
8390
8395
8400
8405
8410
8415
8420
8425
8430
8435
8440
8445
8450
8455
8460
8465
8470
8475
8480
8485
8490
8495
8500
8505
8510
8515
8520
8525
8530
8535
8540
8545
8550
8555
8560
8565
8570
8575
8580
8585
8590
8595
8600
8605
8610
8615
8620
8625
8630
8635
8640
8645
8650
8655
8660
8665
8670
8675
8680
8685
8690
8695
8700
8705
8710
8715
8720
8725
8730
8735
8740
8745
8750
8755
8760
8765
8770
8775
8780
8785
8790
8795
8800
8805
8810
8815
8820
8825
8830
8835
8840
8845
8850
8855
8860
8865
8870
8875
8880
8885
8890
8895
8900
8905
8910
8915
8920
8925
8930
8935
8940
8945
8950
8955
8960
8965
8970
8975
8980
8985
8990
8995
9000
9005
9010
9015
9020
9025
9030
9035
9040
9045
9050
9055
9060
9065
9070
9075
9080
9085
9090
9095
9100
9105
9110
9115
9120
9125
9130
9135
9140
9145
9150
9155
9160
9165
9170
9175
9180
9185
9190
9195
9200
9205
9210
9215
9220
9225
9230
9235
9240
9245
9250
9255
9260
9265
9270
9275
9280
9285
9290
9295
9300
9305
9310
9315
9320
9325
9330
9335
9340
9345
9350
9355
9360
9365
9370
9375
9380
9385
9390
9395
9400
9405
9410
9415
9420
9425
9430
9435
9440
9445
9450
9455
9460
9465
9470
9475
9480
9485
9490
9495
9500
9505
9510
9515
9520
9525
9530
9535
9540
9545
9550
9555
9560
9565
9570
9575
9580
9585
9590
9595
9600
9605
9610
9615
9620
9625
9630
9635
9640
9645
9650
9655
9660
9665
9670
9675
9680
9685
9690
9695
9700
9705
9710
9715
9720
9725
9730
9735
9740
9745
9750
9755
9760
9765
9770
9775
9780
9785
9790
9795
9800
9805
9810
9815
9820
9825
9830
9835
9840
9845
9850
9855
9860
9865
9870
9875
9880
9885
9890
9895
9900
9905
9910
9915
9920
9925
9930
9935
9940
9945
9950
9955
9960
9965
9970
9975
9980
9985
9990
9995
10000
10005
10010
10015
10020
10025
10030
10035
10040
10045
10050
10055
10060
10065
10070
10075
10080
10085
10090
10095
10100
10105
10110
10115
10120
10125
10130
10135

reads programming language source code as input, but which generates executable code for the host platform. The implication of this scenario is that the resulting executable code must be launched on a host-platform, not on a chip platform.

Figure 11 shows the result of compiling the ANSI C programming language source code example of figure 10 using the chip (QX token) compiler shown in figure 8. The code in Figure 11 is also according to the ANSI C programming language. The marked blocks have been compiled into (virtual) machine code for the target platform, then (in one embodiment) encrypted. The QXCode array defined at the end of the code in Figure 11 contains these encrypted virtual machine code for the smart card operating system virtual machine. The marked blocks, and/or function calls to marked blocks containing a function, have been replaced by an anonymous API function call, which serves the purpose of relaying the (encrypted, virtual) machine code blocks to the destination platform, for subsequent storage and execution on the remote platform. The inserted function QXExecutePtr() causes, when executed, all its arguments to be relayed to the smart card chip. These arguments are, amongst other, the QXCode array containing the encrypted virtual machine code.

A significant advantage of using QX tags to mark code and let a compiler replace its contents with QXExecutePtr() style functions, is to alleviate the need for developers to program communication protocols. The task of programming for multiple processors is thereby greatly simplified.

The code in Figure 11 may now be compiled using a commercially available compiler, e.g. Microsoft Visual C++ (indicated by the "Compile for host" arrow in Figure 8), to obtain executable code for a Windows PC host platform (i.e. the "Host and chip executable code" shown in Figure 8).

Notice that the mark-up tags (QXBegin-QXEnd) are enclosed within comments in Figure 10. This allows the code to be compiled either for the host alone (by invoking only the host application compiler), or for the host and chip (by invoking the chip application compiler before the host application compiler). Ordinary compilers ignore comment fields. However, the chip compiler is constructed to be capable of parsing and understanding comment fields of a programming language. The advantage of hiding tags within comment fields is to allow the original (tagged) source code file(s) to be directly compilable by the "final" compiler in the chain (e.g. Microsoft Visual C++ in the example above). This standard compile-scheme

is illustrated in Figure 9. The advantage of allowing a direct compilation of all code for one platform, is to allow simulation and thereby facilitate testing and debugging of the application on just one platform. Now the developer does not need to learn new tools for debugging; but may use the debugging tools which he/she is already familiar with. If tags were not hidden, the source code would not be directly compilable by e.g. Microsoft Visual C++.

In addition to keywords to mark what code belongs to what platform, other keywords can be defined for various purposes: As an alternative to QXBegin QXEnd style marks, a singular mark (e.g. "QX") can be defined to associate a single statement with a specific platform. Another mark "QXUpload" can instruct the compiler to include source code to the purpose of causing one, several or all QX blocks to be uploaded onto the device, without causing any of the QX blocks to be actually invoked. In the case that the execution of QX blocks is controlled by a software license or a capability, marks can be inserted to the effect of affecting the license or capability, e.g. incrementing a counter or setting a timestamp. Keywords may also contain arguments, e.g. QXBegin (PlatformId, LicenseId, AccessMode, Countdown). Here, the compiler may generate code to a) enforce the code to be executed on a specific, named platform, b) enforce conditional execution according to the capabilities given by a specific license, c) Supply an access mode to be verified against a license access mode, also allowing conditional execution, and d) changing the value of a counter.

In one embodiment of the present invention, execution of an application on a platform is controlled by a license residing securely within that platform. A license is equivalent to a capability. It contains attributes which define under what circumstances the applet may be uploaded to the card and/or executed. A license contains different attributes. A unique LicenseId associates uniquely with a specific applet. Other attributes include license limitations (from date/time, to date/time, number of executions, access modes for execution of groups of QX blocks), cryptographic keys, password/pin codes, various text description fields and so on. A license may for instance specify that QX blocks 1 and 4 of the associated applet may execute any number of times until 30. November. QX blocks 2 and 3 of the applet may not execute, and when the license expires on the 30. November, the card holder needs to acquire a new license to renew his rights to execute the applet.

Licenses can be moved from one token to another (provided the license issuer has set the attribute which allows the license to be moved). One token may typically be connected to an online (web) server, or the token functionality may be integrated with the web server itself. This allows other tokens to connect towards the server token and to acquire/purchase new licenses online. Since every license transfer takes place using secure cryptographic protocols, the confidentiality, integrity and license piracy issues are properly taken care of. Such a secure and convenient means for distributing licenses online, along with a secure and convenient means for distributing protected software applications online, facilitates new business opportunities for software rental.

A cryptographic protocol used for the secure license transfer between tokens is described in Figure 12. A license is transferred from token A to token B. The protocol ensures that the license is moved no more than once to the destination token, and never at all to any other token.

Letting token A act as a server token allows licenses to be distributed/checked out to client tokens, as described above. Letting a server token also act in the role of token B allows unused or partially used licenses to be returned to the server. This feature is useful e.g. in corporate license servers to allow licenses to be checked out and in, and thereby float from client to client where the license is required.

In Figure 12, tokens A and B initially share the same (token issuer) transport license. The tokens may also share a vendor transport license. All transport licenses contain a symmetric key. Token A also holds the License which we want to move to token B. If the tokens do not share a common vendor transport license, the issuer transport license is used in its place. The protocol operates as follows: Token B generates a license request including a random transport code in step 1. In step 2 the license request is sent to token A. The license request is integrity protected. In step 3 token A verifies the integrity of the license request. If OK, the token proceeds to generate a session key in step 4. The license is prepared for transfer, the session key is encrypted with the vendor transport license as specified by the license request and the license is deleted from the license store in step 5. In step 6 the license is transmitted to destination token B. Some license attributes are confidentiality protected, notably the license cryptographic key, while the

entire license is integrity-protected. In step 7 the transport codes are compared. This crucial step ensures that a license is moved once and only once to the right token, and never at all to any other token. If the transport codes match, token B replaces the license request (which contained the transport code) with the actual
5 license in step 8.

Changes can be made to the embodiments described above without departing from the broad inventive concept thereof. The present invention is thus not limited to the particular embodiments disclosed, but is intended to cover modifications within the spirit and scope of the present invention.